2017

Randomness from space

Justamante, David

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

RANDOMNESS FROM SPACE

by

David Justamante

March 2017

Thesis Co-Advisors:	George W. Dinolt
	Pantelimon Stănică
Second Reader:	Peter Ateshian

Approved for public release. Distribution is unlimited.

Reissued 30 May 2017 with correction to degree on title page.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2017	Master's Thesis 09-01-2015 to 03-15-2017

4. TITLE AND SUBTITLE

RANDOMNESS FROM SPACE

5. FUNDING NUMBERS

6. AUTHOR(S)

David Justamante

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Randomness is at the heart of today's computing. There are two categorical methods to generate random numbers: pseudorandom number generation (PRNG) methods and true random number generation (TRNG) methods. While PRNGs operate orders of magnitude faster than TRNGs, the strength of PRNGs lies in their initial seed. TRNGs can function to generate such a seed. This thesis will focus on studying the feasibility of using the next generation Naval Postgraduate School Femto Satellite (NPSFS) as a TRNG. The hardware for the next generation will come from the Intel Quark D2000 along with its onboard BMC150 6-axis eCompass. We simulated 3-dimensional motion to see if any raw data from the BMC150 could be used as an entropy source for random number generation. We studied various "schemes" on how to select and output specific data bits to determine if more entropy and increased bitrate could be reached. Data collected in this thesis suggests that the BMC150 contains certain bits that could be considered good sources of entropy. Various schemes further utilized these bits to yield a strong entropy source with higher bitrate. We propose the NPSFS be studied further to find other sources of entropy. We also propose a prototype be sent into space for experimental verification of these results.

14. SUBJECT TERMS

random, randomness, entropy, RNG, space, NPSFS

15. NUMBER OF PAGES 71

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

RANDOMNESS FROM SPACE

David Justamante
Lieutenant, United States Navy
B.S. Computer Science, Liberty University, 2001
B.S. Mathematics, Liberty University, 2001

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN COMPUTER SCIENCE
and
MASTER OF SCIENCE IN APPLIED MATHEMATICS
from the
NAVAL POSTGRADUATE SCHOOL
March 2017

Approved by: George W. Dinolt
Thesis Co-Advisor

Pantelimon Stănică
Thesis Co-Advisor

Peter Ateshian
Second Reader

Peter J. Denning
Chair, Department of Computer Science

Craig W. Rasmussen
Chair, Department of Applied Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Randomness is at the heart of today's computing. There are two categorical methods to generate random numbers: pseudorandom number generation (PRNG) methods and true random number generation (TRNG) methods. While PRNGs operate orders of magnitude faster than TRNGs, the strength of PRNGs lies in their initial seed. TRNGs can function to generate such a seed. This thesis will focus on studying the feasibility of using the next generation Naval Postgraduate School Femto Satellite (NPSFS) as a TRNG. The hardware for the next generation will come from the Intel Quark D2000 along with its onboard BMC150 6-axis eCompass. We simulated 3-dimensional motion to see if any raw data from the BMC150 could be used as an entropy source for random number generation. We studied various "schemes" on how to select and output specific data bits to determine if more entropy and increased bitrate could be reached. Data collected in this thesis suggests that the BMC150 contains certain bits that could be considered good sources of entropy. Various schemes further utilized these bits to yield a strong entropy source with higher bitrate. We propose the NPSFS be studied further to find other sources of entropy. We also propose a prototype be sent into space for experimental verification of these results.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Acronyms and Abbreviations

BBS	Blum-Blum-Shub Pseudorandom Number Generator
CSPRNG	Cryptographically-Secure Pseudorandom Number Generator
IMU	Inertial Measurement Unit
ISSM	Intel System Studio Microcontroller
LSB	Least Significant Bit
MRB	Most Random Bit
MSB	Most Significant Bit
NIST	National Institute of Standards and Technology
NPS	Naval Postgraduate School
NPSFS	NPS Femto Satellite
RF	Radio Frequency
SLSB	Second Least Most Random Bit
STS	Statistical Test Suite
TRNG	True Random Number Generator
UART	Universal Asynchronous Receiver/Transmitter

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

Randomness is at the heart of today's computing. There are two categorical methods to generate random numbers: pseudorandom number generation (PRNG) methods and true random number generation (TRNG) methods. While PRNGs operate orders of magnitude faster than TRNGs, the strength of PRNGs lies in their initial seed. TRNGs can function to generate such a seed. This thesis will focus on studying the feasibility of using the next generation Naval Postgraduate School Femto Satellite (NPSFS) as a TRNG. The hardware for the next generation will come from the Intel Quark D2000 along with its onboard BMC150 6-axis eCompass.

It was assumed that the NPSFS would be tumbling in space, so a special three-dimensional (3D) rig was designed to mimic this 3D motion. The prototype was flashed with particular firmware that would send raw data along with metadata to a collection machine via Bluetooth. Thirteen samples of a million raw data entries each were collected while the prototype was in motion. Another thirteen samples of a million raw data entries were also collected with the rig not moving to see if there were any interesting properties inherent in the system.

The collected raw data was analyzed to determine if any particular bits from the BMC150 exhibited good entropy. We studied various "schemes" on how to select and output specific data bits to determine if more entropy and bitrate could be reached. Each resulting bit stream was run through NIST STS suite of randomness tests along with Fourmilab's ENT suite of tests. The bit streams were also compared against well-known PRNG bit streams as a control. The PRNGs used were BBS, FORTUNA, and Unix /dev/urandom.

The results of this thesis suggest that the BMC150 does in fact contain certain bits that could be considered good sources of entropy. Various schemes further utilized these bits to yield a strong entropy source with higher bitrate.

We propose the NPSFS be studied further to find other sources of entropy. We also propose a prototype be sent into space for experimental verification of these results. Additionally, communication methods and protocols, as well as possible uses should also be studied.

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I wish to thank the Lord for allowing me the opportunity to study at NPS with my family in tow.

I wish to thank my wife and family for their support.

I wish to thank Professor Dinolt for the time and attention given to me for this thesis.

I wish to thank Professor Stănică for his support on my thesis and through his wonderful classes.

I wish to thank the NPS Foundation for their wonderful support for this project.

I wish to thank Dr. Maads, Dr. Manchester, Mr. Ateshian, and CDR Martinsen for all their assistance during this project.

I wish to thank the NPS Summer 2016 Internship Team: Andrew Filo, Salvador Ramirez, Elijah Bigham, Jose Ramirez, and Adela Zamora. Their work proved instrumental in preparation for this project.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Randomness is all around us. Humankind has been taking advantage of randomness since time immemorial. Its uses range from trivial decision making methods, like using a coin toss to determine what shirt to wear, to critical uses that safeguard national secrets and protect the lives of warriors around the globe. Without randomness, patterns in our communications would arise and give way to another person being able to read what we wished to keep secret. Without randomness, simulations would be repetitive and would not emulate the real world. Without randomness, "games of chance" would become "games of predictability," and quite honestly, not much fun to play. It is the key to the "strength" of modern computing communications security and the unsung hero at the core of today's computing.

Though seeming to be an intuitive idea, putting your finger on the definition of randomness is not quite so easy. Over the years, many definitions have been given to what qualifies as randomness, but none seem to satisfy everyone. For the purposes of this paper, we will use the definition provided by Professor D. H. Lehmer in 1951. He informally defined randomness, or more specifically a random sequence where "each term is unpredictable to the uninitiated and whose digits pass a certain number of tests traditional with statisticians" [1]. Another definition in the realm of Computing and Information Theory proposed by Chaitin and Kolmogorov, states that "a series of numbers is random if the smallest algorithm capable of specifying it to a computer has about the same number of bits of information as the series itself" [2].

We now need to define a term to denote the quality of "how good" something is at generating random numbers, a quantitative word to describe "how much" randomness a generator produces. The term we use for this is "entropy." Defining entropy is as elusive as the definition of randomness. It has many meanings for different areas of science. For the purposes of this paper, we will define entropy as "the randomness collected by an operating system or application for use in cryptography or other uses that require random data. This randomness is often collected from hardware sources, either pre-existing ones such as

mouse movements or specially provided randomness generators" [3]. In order to provide a metric for these sources, we quantify and measure the entropy of certain sources by various statistical tests. Now that we have established what we understand what is meant by the terms "randomness" and "entropy", we will discuss the two main categories for the methods of generating random numbers: true random number generation (TRNG) and pseudorandom number generation (PRNG).

True random number generation often involves sampling some phenomena in the natural world with the expectation that the outcome is unpredictable. Generating numbers in nature, though, assumes an axiom at some level that the outcomes of these phenomena cannot be predictable. For example, let us say that we use coin flipping as our generator. While we would assume it would be impossible to determine the result of a coin toss, determining the outcome of coin tossing is actually a physics problem that involves many variables. Nitipak Samsen from the Royal College of Art has made coin flipping devices that yield accurately predictable results [4]. One method that is used to generate random numbers uses atmospheric noise as an entropy source [5], while another well-known method is to use timing of emissions from a radioactive decaying isotopes [6]. Yet another more mundane source involves human-based input, such as computer mouse movement or keyboard typing timing [7].

All of these methods, however, have two main limitations: they cannot be reproduced and it takes a considerable amount of time to acquire a large amount of numbers.

In order to take advantage of today's computing speed, certain deterministic algorithms are implemented on computers to produce a seemingly random sequence, making them pseudorandom instead of truly random. A problem with this method is that these sequences will eventually repeat themselves, although modern algorithms are able to generate number sequences whose period is too long to be practically noticed. In order to be able to reproduce sequences, we start generating numbers based on the same initial conditions. This starting number is often called a "seed" or "initialization vector." The ability to determine the sequence generated from any PRNG algorithm is directly related to the ability to determine the value of the seed. Therefore, the strength of the algorithm is based on the infeasibility of determining the actual value of the seed. In the case that a particular algorithm is created and exhibits enough randomness, that algorithm can be used for cryptographic purposes.

This particular subset of pseudorandom number generators fall into a special subclass called cryptographically-secure pseudorandom number generators (CSPRNGs).

The Naval Postgraduate School Foundation has sponsored research of the different uses of their sponsored Femto Satellite, with true random number generation being one of their interests. Our research will focus on the feasibility of using a Naval Postgraduate School Femto Satellite (NPSFS), or more specifically, its onboard sensors, as a TRNG.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Previous Work

2.1 Random Number History

It is important to survey previous efforts at true random number generation to get insight into how true random numbers are currently being generated. While there are a number of true random number generation setups in existence, we will focus on some of the more popular methods available to the public: radio frequency noise sampling, radioactive isotope emission timing, and computer hardware timing.

2.1.1 Random.org

In 1997, Dr. Mads Haahr set about to build an engine for online gambling. This would require a good random number source to avoid unfair gambling practices. While he eventually moved away from the gambling engine focus, his interest in random number generation continued. He would eventually set up a system to sample radio frequency noise as an entropy source. What has made his work of such value is that this setup is made available to the public by way of the website random.org [5].

2.1.2 Hotbits

Another widely available source of truly random numbers was developed by John Walker, the founder of Autodesk, Inc., by way of his Hotbits program. This setup is based around the unpredictability of radioactive isotope emission timing as the source of entropy. A Geiger-Müller detector tube is interfaced with a computer to detect the emissions in order to produce a random number. He also provides random numbers to the world via his website [6].

2.1.3 Computer Provided Randomness

The importance of random numbers in computing has been known for decades, as noted by Knuth [8]. In 1994, a Linux kernel programmer, Theodore Ts'o, "wrote a driver for Linux,

which takes information about hard [difficult] to predict events like keyboard and mouse use, packet and disk drive timings, and so on, and used it to seed a cryptographically secure random number generator" [9]. Since Ts'o's implementation, other modules have been implemented to address some of the algorithm's weaknesses and limitations. However, the popularity and importance of this module has resulted in this module and its follow-on implementations to be integrated in most major operating systems [9].

2.2 Small Satellite Origins

We now turn our attention to the historical milestones in the lead up to the NPSFS. We will discuss the origins of the KickSat project, the NPS Foundation involvement.

2.2.1 KickSat

Zachary Manchester, a graduate student in Aerospace Engineering at Cornell University [10] stood up a Kickstarter [11] campaign for KickSat (see Figure 2.1). The KickSat initiative was to design, build and deploy small, inexpensive, consumer-grade satellites. Because of their small size, they can be loaded into a launcher with a very small footprint. Once the launcher is loaded, the launcher is set into space, where the launcher ejects the small satellites into orbit (see Figure 2.3). Because of their low cost, many of these satellites could be made and deployed and be programmed for various uses. The result was an inexpensive satellite network that could be deployed on a minimal budget.

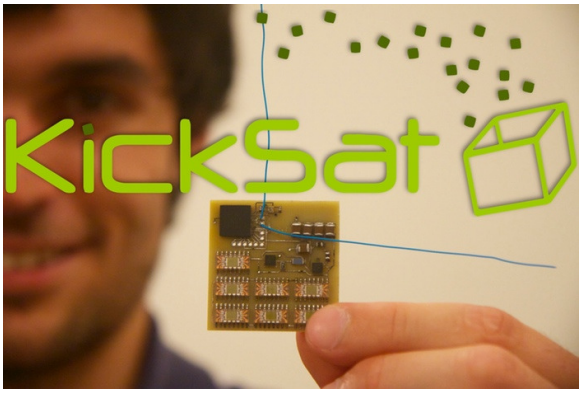

Figure 2.1. KickSat. Source: [10].
Photo credit: Zachary Manchester

2.3 NPS Foundation Sponsorship

The NPS Foundation decided to sponsor NPS research and further develop the idea of using Femto Satellites-size platforms for DoD purposes. This is the origin of the NPS Femto Satellite (NPSFS). Research began both on component selection for the generations of satellites as well as viable functions of individual satellites (see Figure 2.2).

Some of the research areas are: using a satellite network for geolocation in a GPS denied environment, spreading them on the surface of the ocean to act as a relay between underwater and overhead communications assets, an ad-hoc communications network for on-demand communication needs. The research area that was chosen for this specific thesis is the ability to use the NPSFS as a True Random Number Generator.

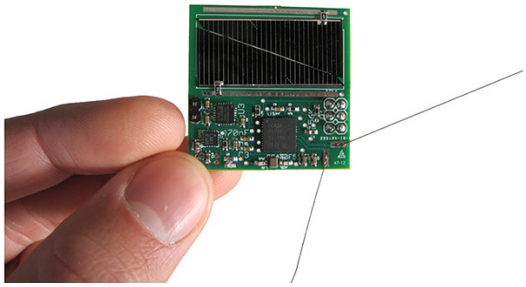

Figure 2.2. NPSFS. Source: [12]. Photo credit: Andy Filo

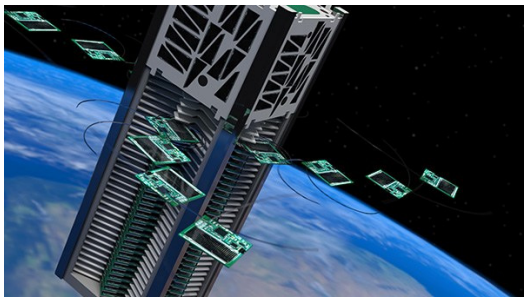

Figure 2.3. NPSFS Launcher. Source: [12]. Photo credit: Ben Bishop

2.4 NPSFS TRNG

While some of the TRNGs mentioned require special equipment, these are only available to the immediate audience, or require access to an infrastructure. Our work will seek to be able to provide truly-random numbers amongst a satellite network cluster. We will study how good of an entropy sources would the specific sensor on the NPSFS be at producing random numbers, discuss various constraints and assumptions, explain our methodology, present the results of our studies, and finally wrap up discussing our outcome and recommend some other studies that would help further research in this topic.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Study

In order to study the ability of the NPSFS to generate truly-random numbers, we first decided what the future NPSFS components will be. Once they were identified, we determined which component we would use as an entropy source. We then programmed the device to output raw data. After that, we simulated the expected environment the NPSFS will be in. Once under simulated conditions, raw data from the device was gathered. Once gathered, the data was tested to see if we are able to extract random numbers from this data. We then analyzed our results.

3.1 Hardware Selection

The Electrical and Computer Engineering Department at NPS hosted a college internship program over the summer of 2016 [13]. These interns were led by NPS Professor Peter Ateshian. At the end of the summer program, the group had selected the Intel Quark processor as the planned processor for future NPSFS outfitting. The Quark process was selected due to its extremely low power usage, reasonable speed, and ability to do certain mathematical calculations in few cycles.

The development kit that was used for testing the Intel Quark processor was the Intel Quark D2000 development kit (see Figure 3.1). The D2000 was designed with Arduino-compatible header layout for use with existing Arduino shields. The D2000 also came with a battery bay that would allow the D2000 to be powered with a coin-cell battery. Finally, the D2000 came with a 6-axis eCompass, the Bosch BMC150. This eCompass would provide 3-axis accelerometer and 3-axis magnetometer. This eCompass was also chosen as a component to be put on the future NPSFS generations.

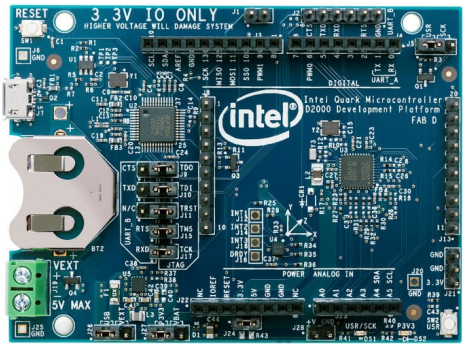

Figure 3.1. Intel Quark D2000. Source: [14].

3.2 Entropy Source Identification

The NPSFS will be outfitted with a number of different components that could be potential sources of entropy for random number generation. Some of the possible candidates are as follows: unshielded memory subject to cosmic radiation that might flip bits, location and timing information; on-board radio transceiver sampling radio frequency noise, similar to Dr. Haahr's experiment, but implemented on a Femto satellite in space; the on-board inertial measurement unit (IMU). An IMU provides data about the physical orientation and motion of the device. An IMU provides accelerometer data, measuring how the device is accelerating or decelerating in all three axes, as well as magnetometer data, which measures there the device is "pointing" relative to the earth's magnetic field, like a normal compass except proving you with three dimensional orientation data. The Bosch BMC150 was selected as the IMU that would used on the upcoming generations of NPSFS. Since it was already available on the D2000, we studied how to generate truly-random numbers with the D2000 and on-board BMC150.

3.3 Firmware Programming

Once we agreed that the BMC150 was the device that was to be studied, we needed to get the raw data from the device registers. The software Intel provided to program the D2000 was the Intel System Studio Microcontrollers. This suite came with example programs using the accelerometer and magnetometer to demonstrate interfacing and using the data from the BMC150. Studying the datasheet for the BMC150 [15], we found that the BMC150 stores

its accelerometer data, magnetometer data, and temperature data in three different register banks in its internal memory. Preliminary studies showed that the temperature did not have sufficient resolution to experience minute temperature changes and was therefore not considered for this study. We investigated the particular register layouts of the accelerometer and magnetometer register banks to get a feel of what data we had at our disposal.

3.3.1 Layout of BMC Registers

As laid out in the BMC150 data sheet, all three accelerometer registers are 12 bits long, followed by three empty bits, ending with a "new data" bit (see Figure 3.2). This "new data" bit is 1 if that particular register contains a fresh value that has not been "seen" by the D2000. Otherwise, a 0 would indicate that this register contains a "stale" value that has already been "seen" by the D2000 and not refreshed with a new value. For this reason, good raw register entries should be odd. If they were even, with the last bit would being a 0, this would indicate a stale entry that has already been seen.

		Bit Positions															
		0xF	0xE	0xD	0xC	0xB	0xA	0x9	0x8	0x7	0x6	0x5	0x4	0x3	0x2	0x1	0x0
Accelerometer Axis Registers	X	Bit_11	Bit_10	Bit_9	Bit_8	Bit_7	Bit_6	Bit_5	Bit_4	Bit_3	Bit_2	Bit_1	Bit_0	blank	blank	blank	new_data_X
	Y	Bit_11	Bit_10	Bit_9	Bit_8	Bit_7	Bit_6	Bit_5	Bit_4	Bit_3	Bit_2	Bit_1	Bit_0	blank	blank	blank	new_data_Y
	Z	Bit_11	Bit_10	Bit_9	Bit_8	Bit_7	Bit_6	Bit_5	Bit_4	Bit_3	Bit_2	Bit_1	Bit_0	blank	blank	blank	new_data_Z

Figure 3.2. Layout of Accelerometer Registers

The Magnetometer register in the Z direction is 15 bits long, followed by its "new data" bit (see Figure 3.3). The magnetometer in the X and Y directions are both 13 bits long, followed by two empty bits, and end with their respective "new data" bits. These "new data" bits behave like the "new data" bits in the accelerometer registers.

		Bit Positions															
		0xF	0xE	0xD	0xC	0xB	0xA	0x9	0x8	0x7	0x6	0x5	0x4	0x3	0x2	0x1	0x0
Magnetometer Axis Registers	X	Bit_12	Bit_11	Bit_10	Bit_9	Bit_8	Bit_7	Bit_6	Bit_5	Bit_4	Bit_3	Bit_2	Bit_1	Bit_0	Fixed '0'	Fixed '0'	X_self_test
	Y	Bit_12	Bit_11	Bit_10	Bit_9	Bit_8	Bit_7	Bit_6	Bit_5	Bit_4	Bit_3	Bit_2	Bit_1	Bit_0	Fixed '0'	Fixed '0'	Y_self_test
	Z	Bit_14	Bit_13	Bit_12	Bit_11	Bit_10	Bit_9	Bit_8	Bit_7	Bit_6	Bit_5	Bit_4	Bit_3	Bit_2	Bit_1	Bit_0	Z_self_test

Figure 3.3. Layout of Magnetometer Registers

3.3.2 Programming the Firmware

Once the layout of the particular registers was determined, we set about writing the code to be flashed onto the D2000. We tweaked the BMC150 library code to allow access to the raw register data and wrote a small program that combined the reading of the accelerometer and magnetometer. This program outputs the raw data in hex format as a comma-separated list having the three accelerometer axes data, the three magnetometer axes data on a single line out to the Universal Asynchronous Receiver/Transmitter (UART) serial interface on the board.

3.4 Environment Simulation

Once we understood how to output raw data over the serial interface, we needed to simulate the environment the device would be operating in. We did not know how the device would tumble in space. Based on this, we assumed that the device would be moving in a random three-dimensional (3D) motion. To simulate this, a special rig was designed and fabricated (see Figure 3.4). This rig was designed to spin in two axes, which would simulate motion in 3 axes. One controller was used to control the speed of the motor that spun the U-shaped arm. The other controller was connected to a slip ring in the base to control the speed of the motor that would spin the D2000 enclosure along the spinning arm's axis. To minimize cumbersome connections, we decided to use an HC-06 Bluetooth module to communicate the serial data from the D2000 to our collection machine. Instead of having another set of slip rings pass power to the device, we decided to power the D2000 along with its Bluetooth device using inexpensive hobby LiPo batteries. This choice was made because a 1-cell LiPo provides the 3.3V needed for D2000 and Bluetooth to operate and weighs very little and minimizes the effect on the spinning motion.

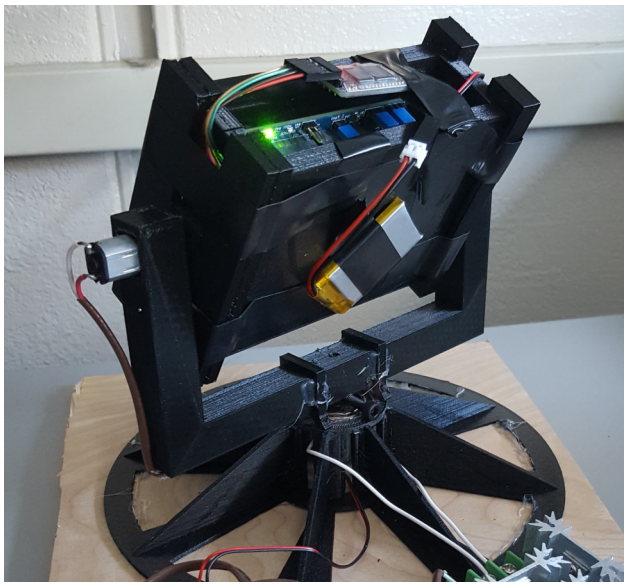

Figure 3.4. 3D Motion Simulation Rig

3.5 Data Gathering

Once the device was paired with the collection machine, we started to collect data to see if there were any idiosyncrasies in collecting the data. We collected one million samples while the device was stationary and one million samples while the device was moving. The amount of time it took to collected one million samples from the stationary setup was approximately 45 minutes, while the time it took to collect the one million samples from the moving setup was more than 2 hours. In order to determine if the delay was caused by loss of data or if the device was simply taking longer to transmit the data, we added an incrementing counter to act as a local serial number and a D2000 tick counter to the end of the raw data line broadcast by the D2000. A small program was coded up to take the time stamp of when the particular data was received and pre-pended it to the received raw data [Appendix A]. We also wanted to identify if Bluetooth was causing problems, so we split the D2000 UART connection to output to both the Bluetooth module as well as a serial cable to the collection machine. One million samples were run and collected simultaneously. The result showed no excessive differences in the data collected from the Bluetooth and the direct serial cable, so we concluded that the delay was caused by the device providing data a bit slower than anticipated which would not severely impact our study.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Methodology

We now get to the task of understanding and working with a whole lot of data. We will first describe the amount and kinds of data we collected. We follow this by describing the anatomy of a raw data entry. Then, we will discuss the different ways we handled the raw data in order to extract numbers that prove to be random. We will then briefly discuss some pseudorandom number generated data as a control. We will then discuss the various tests that were selected in order to determine if these numbers are random or not.

4.1 Data Sets

Because we are testing a number generator that is constrained to the physical world, generating these raw data sets took hours to produce one usable raw data set. In some instances, the rig was left to generate numbers, only to return and find that particular dataset to be faulty, due to the rig getting stuck. In the end, we were able to collect thirteen datasets with the rig in motion. Each dataset consisted of one million entries. We also were able to gather thirteen datasets with the rig not in motion. These data sets also consisted of one million entries each.

4.2 Data Entry Description

Each entry in these raw datasets has the same anatomy. The first entry is a timestamp placed by the collecting machine to track what time the particular entry was received. This timestamp is expressed as the number of milliseconds since January 1, 1970, 12:00 am. The timestamp is followed by a semicolon to mark the beginning of data received from the D2000.

Unfortunately, the output function available to the D2000 from the programming suite only allows a limited subset of "printf" functionality [16]. In attempt to save space, all the values sent are in hexadecimal format and separated by a comma. Each register is 16-bits long, though the output might suggest more bits due to the two's complement representation of negative values. The first six values output from the D2000 are, in order: accelerometer

X, accelerometer Y, accelerometer Z, magnetometer X, magnetometer Y, magnetometer Z. After these values, the next value is the local serial number in hexadecimal format. The final value output is the tick counter on the D2000 in hexadecimal format (see Figure 4.1).

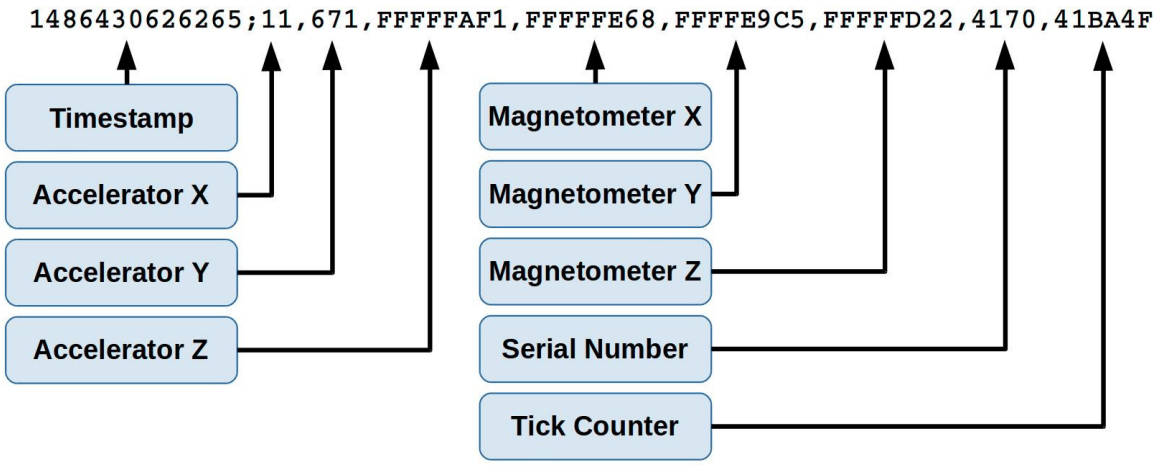

Figure 4.1. Explanation of Raw Data Entry

4.3 Schemes

Now that we had a good number of raw datasets, our goal was to see if we can use this raw data in some way to extract random numbers from it. For the purposes of this paper, we used the term "scheme" to describe a specific way of combining specific bits in order to see if the resulting output from a particular scheme was random. We determined if they were random by running this output sequence through a number of widely-used tests to determine if a particular bit sequence was random. For the remaining subsections of this chapter, we introduce a particular notation to help clarify what exactly we are referring to. The notation will consist of a letter A for an accelerometer register bank or M for a magnetometer register bank. We follow that first character by the specific axis in that register we are looking at: X for the X-axis, Y for the Y-axis, and Z for the Z-axis. We follow this with a hexadecimal value referencing the specific bit in that register on which we are focusing. For example, **AY5** is the accelerometer Y-axis register's 0x5 bit; **MZC** is the magnetometer Z-axis 0xC bit. For a visual reference, please refer to Figures 3.2 and 3.3.

4.3.1 Scheme 1 - Individual Bits

This scheme is the most straight forward. We will focus on a particular bit in a particular register. We will take that value in each entry of the raw data and output them one after another. This will enable us to identify if any particular bit is a good entropy source in and of itself. Here is the pseudocode and visual representation (Figure 4.2) of the Individual Bits scheme:

Scheme 1 Algorithm

```
1: procedure SCHEME1(head, register, field)
2:     entry ← *head                    ▷ Start at the beginning of the entry list
3:     while entry ≠ null do            ▷ While not the end of entry list
4:         output ← entry[register][bit]   ▷ Output specific bit
5:         entry ← entry.next           ▷ Get the next entry
```

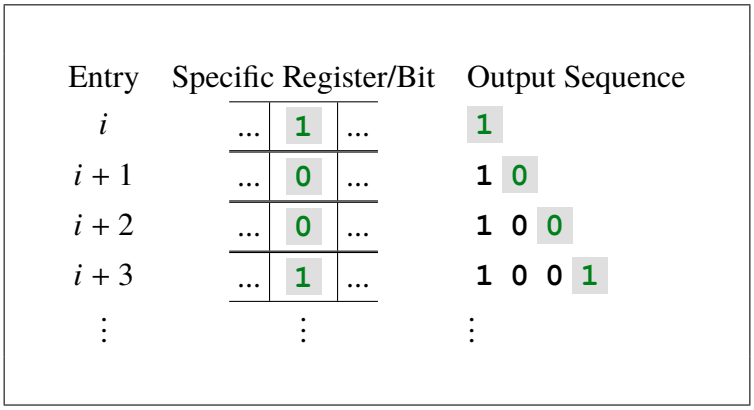

Figure 4.2. Individual Bits

Things to note for upcoming schemes

We will show in the next chapter the results of this experiment, but we will move forward with the following knowledge. **AX4** is the bit that seems to be the best entropy source in the accelerometer X register. We will call this bit the most random bit (MRB) in accelerometer X. **AY5** is the bit that seems to be the best entropy source in the accelerometer Y register. We will call this bit the MRB of accelerometer Y. **AZ5** is the bit that seems to be the best entropy source in the accelerometer Z register. We will call this bit the MRB of accelerometer Z. Since none of the magnetometer registers appear to be good entropy sources, when we need a particular bit to be used from the magnetometer registers, we will use the least

significant data bit (LSB) from each register. For magnetometer X, the LSB is **MX3**. For magnetometer Y, the LSB is **MY3**. For magnetometer Z, the LSB is **MZ1**. In some future schemes, we will want to use other bits from the accelerometer registers in addition to the MRB. We selected **AX5** to be the second most random bit (SMRB) in the accelerometer X register, **AY6** as the SMRB in the accelerometer Y register, and **AZ6** as the SMRB in the accelerometer Z register. Armed with these bits, we can then use this information to configure the following schemes.

4.3.2 Scheme 2 - Accelerometer X/Y/Z MRB Concatenation

In this scheme, we will take the MRB from each of the three accelerometer registers and concatenate them before sending them as output. For clarification, we will be putting them in the following order: **AX4**, **AY5**, **AZ5**. Here is the pseudocode and visual representation (Figure 4.3) of the accelerometer X/Y/Z MRB Concatenation scheme:

Scheme 2 Algorithm

```
1: procedure SCHEME2(head)
2:     entry ← *head                    ▷ Start at the beginning of the entry list
3:     while entry ≠ null do            ▷ While not the end of entry list
4:         output ← entry[Acc_X][MRB]   ▷ Output Acc_X MRB
5:         output ← entry[Acc_X][MRB]   ▷ Output Acc_Y MRB
6:         output ← entry[Acc_X][MRB]   ▷ Output Acc_Z MRB
7:         entry ← entry.next           ▷ Get the next entry
```

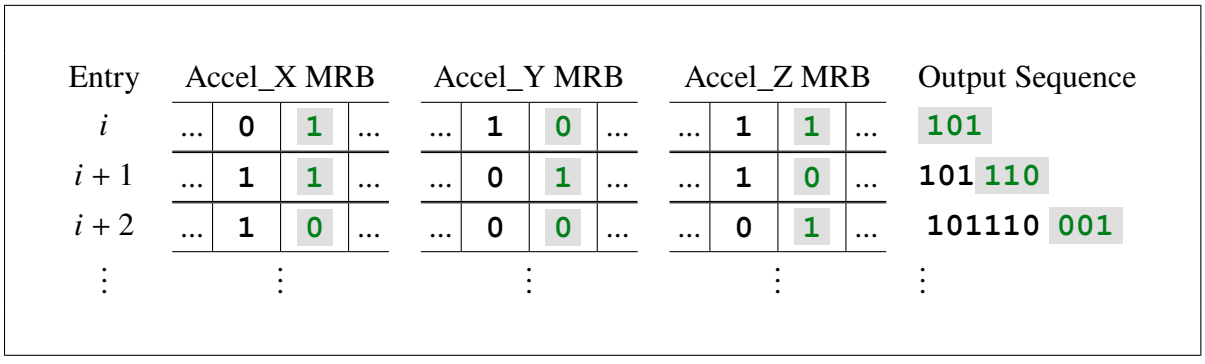

Figure 4.3. Concatenating Accelerometer LSB

4.3.3 Scheme 3 - Accelerometer MRB Shrinking Using Magnetometer LSB Trigger

For this scheme, we used a concept that was published in 1993 by Don Coppersmith, Hugo Krawczyk, and Yishay Mansour. [17] A short explanation of this scheme is the use of one piece of data to determine if another piece of data is output. For the purposes of this scheme, we will use a magnetometer axis LSB to trigger the corresponding axis' accelerometer MRB to be output. If **MX3** is a 1, **AX4** will be output, but if **MX3** is a 0, **AX4** will not be output. Similarly if **MY3** is a 1, **AY5** will be output, but if **MY3** is a 0, **AY5** will not be output. Finally if **MZ1** is a 1, **AZ5** will be output, but if **MZ1** is a 0, **AZ5** will not be output. Here is the pseudocode and visual representation (Figure 4.4) of the accelerometer MRB Shrinking using magnetometer LSB Trigger scheme:

Scheme 3 Algorithm

```
1:  procedure SCHEME3(head)
2:      entry ← *head                          ▷ Start at the beginning of the entry list
3:      while entry ≠ null do                  ▷ While not the end of entry list
4:          if Mag_XLSB = 1 then
5:              output ← entry[Acc_X][MRB]     ▷ Output Acc_X MRB if Mag_X LSB
6:          if Mag_YLSB = 1 then
7:              output ← entry[Acc_Y][MRB]     ▷ Output Acc_Y MRB if Mag_Y LSB
8:          if Mag_ZLSB = 1 then
9:              output ← entry[Acc_Z][MRB]     ▷ Output Acc_Z MRB if Mag_Z LSB
10:         entry ← entry.next                 ▷ Get the next entry
```

Entry	Accel_X MRB			Mag_X LSB			Output Sequence
i	...	0	1 ...	...	1	0 ...	
$i+1$	...	1	1 ...	...	0	1 ...	1
$i+2$	...	1	0 ...	...	0	0 ...	1
$i+3$	...	1	0 ...	...	0	1 ...	1 0
⋮		⋮			⋮		⋮

Figure 4.4. Using Magnetometer LSB as Shrinking Trigger

4.3.4 Scheme 4 - Accelerometer MRB Shrinking Using Accelerometer SMRB Trigger

This scheme is similar to Scheme 3 with the alteration that the trigger is no longer the LSB of a magnetometer LSB, but rather the second most random bit (SMRB) of the same accelerometer register. If **AX5** is a 1, **AX4** will be output, but if **AX5** is a 0, **AX4** will not be output. Similarly if **AY6** is a 1, **AY5** will be output, but if **AY6** is a 0, **AY5** will not be output. Finally, if **AZ6** is a 1, **AZ5** will be output, but if **AZ6** is a 0, **AZ5** will not be output. Here is the pseudocode and visual representation (Figure 4.5) of the accelerometer MRB Shrinking using accelerometer SMRB Trigger scheme:

Scheme 4 Algorithm

```
1:  procedure SCHEME4(head)
2:      entry ← *head                          ▷ Start at the beginning of the entry list
3:      while entry ≠ null do                  ▷ While not the end of entry list
4:          if Acc_XSMRB = 1 then
5:              output ← entry[Acc_X][MRB]     ▷ Output Acc_X MRB if Acc_X SMRB
6:          if Acc_YSMRB = 1 then
7:              output ← entry[Acc_Y][MRB]     ▷ Output Acc_Y MRB if Acc_Y SMRB
8:          if Acc_ZSMRB = 1 then
9:              output ← entry[Acc_Z][MRB]     ▷ Output Acc_Z MRB if Acc_Z SMRB
10:         entry ← entry.next                 ▷ Get the next entry
```

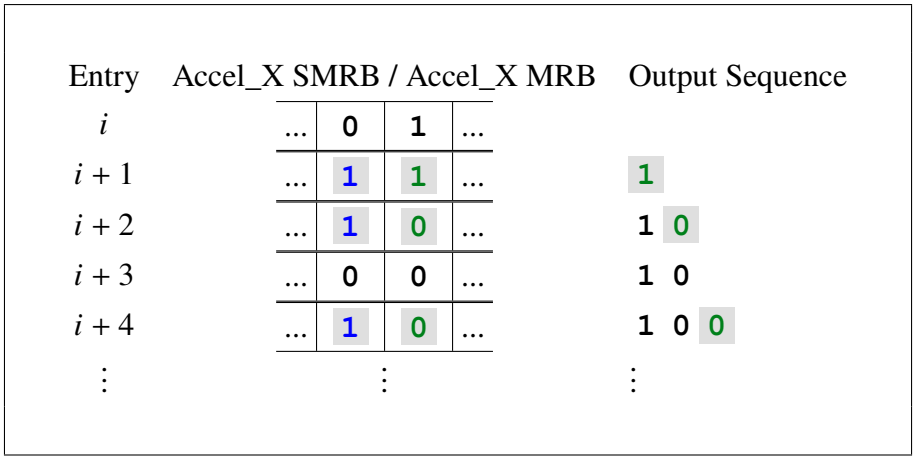

Figure 4.5. Using Accelerometer SLSB as Shrinking Trigger

4.3.5 Scheme 5 - Accelerometer MRB Timed by Magnetometer 2xLSBs

This scheme is somewhat a variation of the Shrinking Generator used in the previous two schemes. In this scheme, we take a MRB from an accelerometer register, then take the value from the two LSBs from the corresponding magnetometer register. The value of these bits will determine how many of the next entries will be skipped before this particular accelerometer MRB will be taken again with the magnetometer two LSBs being taken again. The two LSBs for magnetometer X are **MX3** and **MX4**. The two LSBs for magnetometer Y are **MY3** and **MY4**. The two LSBs for magnetometer Z are **MZ1** and **MZ2**. Here is the pseudocode and visual representation (Figure 4.6) of the accelerometer MRB Timed by magnetometer 2xLSBs scheme:

Scheme 5 Algorithm

```
1:  procedure SCHEME5(head)
2:      entry ← *head                          ▷ Start at the beginning of the entry list
3:      timerX, timerY, timerZ ← 0
4:      while entry ≠ null do                  ▷ While not the end of entry list
5:          if timerX = 0 then
6:              output ← entry[Acc_X][MRB]     ▷ Output Acc_X MRB if timerX is 0
7:              timerX ← Mag_X2xLSB            ▷ timerX gets value from Mag_X 2xLSB
8:          else
9:              timerX = timerX − 1                                  ▷ Decrement timerX
10:         if timerY = 0 then
11:             output ← entry[Acc_Y][MRB]     ▷ Output Acc_Y MRB if timerY is 0
12:             timerY ← Mag_Y2xLSB            ▷ timerY gets value from Mag_Y 2xLSB
13:             timerY = timerY − 1                                  ▷ Decrement timerY
14:         if timerZ = 0 then
15:             output ← entry[Acc_Z][MRB]     ▷ Output Acc_Z MRB if timerZ is 0
16:             timerZ ← Mag_Z2xLSB            ▷ timerZ gets value from Mag_Z 2xLSB
17:         else
18:             timerZ = timerZ − 1                                  ▷ Decrement timerZ
19:         entry ← entry.next                                       ▷ Get the next entry
```

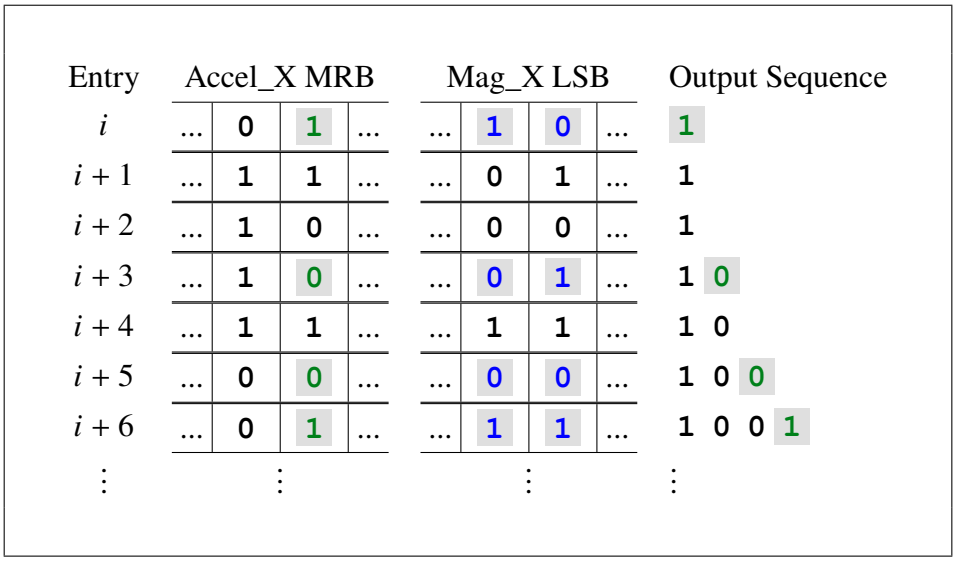

Figure 4.6. Using Magnetometer LSBs as Sleep Timer

4.3.6 Scheme 6 - Accelerometer MRB Timed by Accelerometer 2xSMRBs

This scheme is similar to the previous scheme with the exception that instead of using the two magnetometer LSBs. The two bits starting with accelerometer X SMRBs will be used. These two bits for accelerometer X are **AX5** and **AX6**. The two bits starting with accelerometer Y SMRBs will be used. These two bits for accelerometer Y are **AY6** and **AY7**. the two bits starting with accelerometer Z SMRBs will be used. These two bits for accelerometer Z are **AZ6** and **AZ7**. Here is the pseudocode and visual representation (Figure 4.7) of the accelerometer MRB Timed by accelerometer 2xSMRBs scheme:

Scheme 6 Algorithm

```
1:  procedure SCHEME6(head)
2:      entry ← *head                          ▷ Start at the beginning of the entry list
3:      timerX, timerY, timerZ ← 0
4:      while entry ≠ null do                  ▷ While not the end of entry list
5:          if timerX = 0 then
6:              output ← entry[Acc_X][MRB]     ▷ Output Acc_X MRB if timerX is 0
7:              timerX ← Acc_X2xSMRB           ▷ timerX gets value from Acc_X 2xSMRB
8:          else
9:              timerX = timerX − 1            ▷ Decrement timerX
10:         if timerY = 0 then
11:             output ← entry[Acc_Y][MRB]     ▷ Output Acc_Y MRB if timerY is 0
12:             timerY ← Acc_Y2xSMRB           ▷ timerY gets value from Acc_Y 2xSMRB
13:         else
14:             timerY = timerY − 1            ▷ Decrement timerY
15:         if timerZ = 0 then
16:             output ← entry[Acc_Z][MRB]     ▷ Output Acc_Z MRB if timerZ is 0
17:             timerZ ← Acc_Z2xSMRB           ▷ timerZ gets value from Acc_Z 2xSMRB
18:         else
19:             timerZ = timerZ − 1            ▷ Decrement timerZ
20:         entry ← entry.next                 ▷ Get the next entry
```

Entry	Accel_X 2 SMRB Timer, Accel_X MRB				Output Sequence
i	...	1	0	1	...	1
$i+1$	...	0	1	1	...	1
$i+2$	...	0	0	0	...	1
$i+3$	...	0	1	0	...	1 0
$i+4$	...	1	1	1	...	1 0
$i+5$	...	0	0	0	...	1 0 0
$i+6$	...	1	1	1	...	1 0 0 1
⋮		⋮				⋮

Figure 4.7. Using Accelerometer SLSBs as Sleep Timer

4.3.7 Scheme 7 - Register Parity

This scheme simply computes and takes the parity bit of each of the accelerometer and magnetometer registers and outputs them. The order of output is as follows: accelerometer X parity, accelerometer Y parity, accelerometer Z parity, magnetometer X parity, magnetometer Y parity, and magnetometer Z parity. Here is the pseudocode and visual representation (Figure 4.8) of the Register Parity scheme:

Scheme 7 Algorithm

```
1:  procedure SCHEME7(head)
2:      entry ← *head                          ▷ Start at the beginning of the entry list
3:      while entry ≠ null do                  ▷ While not the end of entry list
4:          output ← parity(entry[Acc_X])      ▷ Output Parity of entire Acc_X register
5:          output ← parity(entry[Acc_Y])      ▷ Output Parity of entire Acc_Y register
6:          output ← parity(entry[Acc_Z])      ▷ Output Parity of entire Acc_Z register
7:          output ← parity(entry[Mag_X])      ▷ Output Parity of entire Mag_X register
8:          output ← parity(entry[Mag_Y])      ▷ Output Parity of entire Mag_Y register
9:          output ← parity(entry[Mag_Z])      ▷ Output Parity of entire Mag_Z register
10:         entry ← entry.next                 ▷ Get the next entry
```

Entry	Register								Parity Bit	Output Sequence
i	0	0	0	0	1	0	1	0	0	0
$i+1$	1	0	0	1	1	0	0	0	1	0 1
$i+2$	0	1	1	0	0	1	1	0	0	0 1 0
$i+3$	1	1	1	0	1	1	0	0	1	0 1 0 1
$i+4$	1	0	1	0	0	0	0	0	0	0 1 0 1 0
$i+5$	1	1	1	1	1	1	0	1	1	0 1 0 1 0 1
⋮				⋮					⋮	⋮

Figure 4.8. Using Parity Bits Of Each Data Source

4.4 Control Sets

In order to see how these bitstreams compare widely-used random number generators, we selected a few PRNGs to test against. We selected the Blum-Blum-Shub (BBS) PRNG

as our first control. The second PRNG we selected was the FORTUNA PRNG. The third PRNG used was the "/dev/urandom" device on the collection machine. For each PRNG, 13 samples of one million bits were collected and run through the same tests as our project data. The code used to generate the BBS datasets was downloaded from https://github.com/graffitiplum/gmpbbs. The code used to generate the FORTUNA datasets was downloaded from http://www.seehuhn.de/pages/fortuna and modified slightly.

4.5 Testing

While there are a number of test suites that are used for testing whether a sequence is random or not, we decided to use two test suites in order to determine if our bit sequences were random. These test suites are: the NIST STS and Fourmilabs' ENT test suite.

4.5.1 NIST STS Tests

The U.S. National Institute of Standards and Technology (NIST) Statistical Test Suite (STS) is widely used as a test suite to determine whether a particular bit sequence is random. Some great work was already been done [18], [19] in explaining, comparing, and contrasting the different STS tests. The current version of STS runs fifteen different tests on a provided data stream and returns a common measure to use for evaluation, a p-value.

Evaluation Criteria

For this research, as suggested by NIST, we decided to set the threshold for any test to be a p-value of .01. If any tests return a value less than .01, we considered that particular bit sequence to have not passed that test. There are a few reasons particular tests are not passed. In some cases, that sequence simply fails to pass the threshold. Another reason a test might not be passed is that it did not apply to that bit sequence, since certain criteria are not being met for when the test was conducted. An example of such a test is the Random Excursion test that will not return successfully if the bit sequence is not long enough. When a test fails to pass, the STS returns a value of zero. Another fact to note is that there are some tests that return multiple p-values. In those cases, the arithmetic mean of those p-values is taken and reported in our results.

4.5.2 Fourmilabs' ENT Suite

Fourmilab in Switzerland has a test suite that conducts five tests. They are: Entropy Test, Chi-squared Test, Mean Test, Monte-Carlo Pi Estimation Test, and Serial Correlation Test. Descriptions of these tests are documented on the Fourmilab Documentation page for this test suite [20]. This test suite brings a collection of tests from different disciplines to determine randomness.

Evaluation Criteria

Unlike the NIST STS, each test in the ENT suite has a unique metric. In order for a particular bit stream to be considered a success, each test must fall within a certain range.

Entropy Test The Entropy test returns how much information per bit there is. For this test to be considered a success, the value must be greater than .995. We note that the maximum amount of information a bit can contain is 1 bit per bit, so 1.00 is the upper end of our test.

Chi-Squared Test The Chi-Squared test returns the Chi-squared value of the particular sequence. For this test to be considered a success, the value must be less than 127.

Mean Test The Mean test returns the ratio of 1's divided by the total amount of bits. This is similar to NIST's Monobit Frequency test. For this test to be considered a success, the value must be within .005 of .500. More specifically, the value must fall between .495 and .505.

Monte-Carlo Value of Pi Test The Monte-Carlo Value of Pi test returns an approximation of Pi. For this test to be considered a success, the value must be within .02 of Pi. More specifically, the value must fall between approximately 3.121 and 3.161.

Serial Correlation Test The Serial Correlation test returns a value denoting how related one bit is to the previous bit. This value ranges between -1 and 1. For this test to be considered a success, the value must be within .003 of 0. More specifically, the value must fall between -.003 and .003.

CHAPTER 5:
Results

We will now discuss the results of running our various bit streams through the NIST STS tests and Fourmilab ENT Suite tests. As we will discuss the numerical test results of random numbers being run through statistical tests, it is an understatement to say we will come across a few numbers. In an effort to present the results in a more communicable manner, we will be using various graphs to represent the summary results as well as the detailed results.

5.1 Interpretation of Results

Overview Results for Scheme 1

The overview of Scheme 1 results is in the form of a grid. This overview graph is summary of Scheme 1 results run through the STS tests. The vertical axis shows the different states (moving and static), subdivided into their accelerometer and magnetometer axis registers. The horizontal axis is the bit position in each of the registers. This format was chosen to be able to survey all the registers against a similar metric, the p-value from the different STS tests. Passing and failing regions are colored in green and red respectively. For further differentiation between the moving and static data sets, successful moving data values are colored solid green and successful static data values are colored solid blue. Results that fail to pass a test, in both moving and static, are colored red.

Summary Results

The summary graphs are in a quadrant format. The top row is the summary of the moving data set and the bottom row is the summary of the static data set. Each row is separated into two sections, the left being the results of the STS test, using the same metric of p-value, and the right being the five different ENT Suite tests.

Detailed Results

In all the detailed graphs, the graph regions are colored in green and red to denote whether a particular result passed or not. We decided to denote a single result. The more the data overlaps, the darker the color would be. The left-most column will show the results of the 13 samples of each PRNG as a control. The blue points represent the Blum-Blum-Shub data streams. The green points represent the /dev/urandom bit streams. The red points represent the FORTUNA data streams. For the remaining columns, each bit has two different data sets, the left purple points denote the data set when the D2000 was moving and the right yellow points which denote the data set when the D2000 was static.

5.2 Scheme 1 Results

The results of Scheme 1, Figure 5.1, show that when moving, the **AX4**, **AY5**, and **AZ5** exhibit good entropy in and of themselves. Other bits that show decent randomness are **AX5**, **AX6**, **AY4**, **AY6**, **AY7**, **AZ4**, and **AZ6**. We note that these honorable mentions either pass all the tests but not as well as **AX4**, **AY5**, and **AZ6**, or they fail only a few tests. The magnetometer bits while moving do not seem to exhibit all that much entropy. For this reason, we selected **AX4**, **AY5**, and **AZ5** to be the most random bits (MRB) and used these specific bits in the schemes that follow. The static data set does not show any particular bit to be a good entropy source. However, **AX4**, **AY4**, and **AZ4** do show some anomalous behavior that would be interesting for a future study.

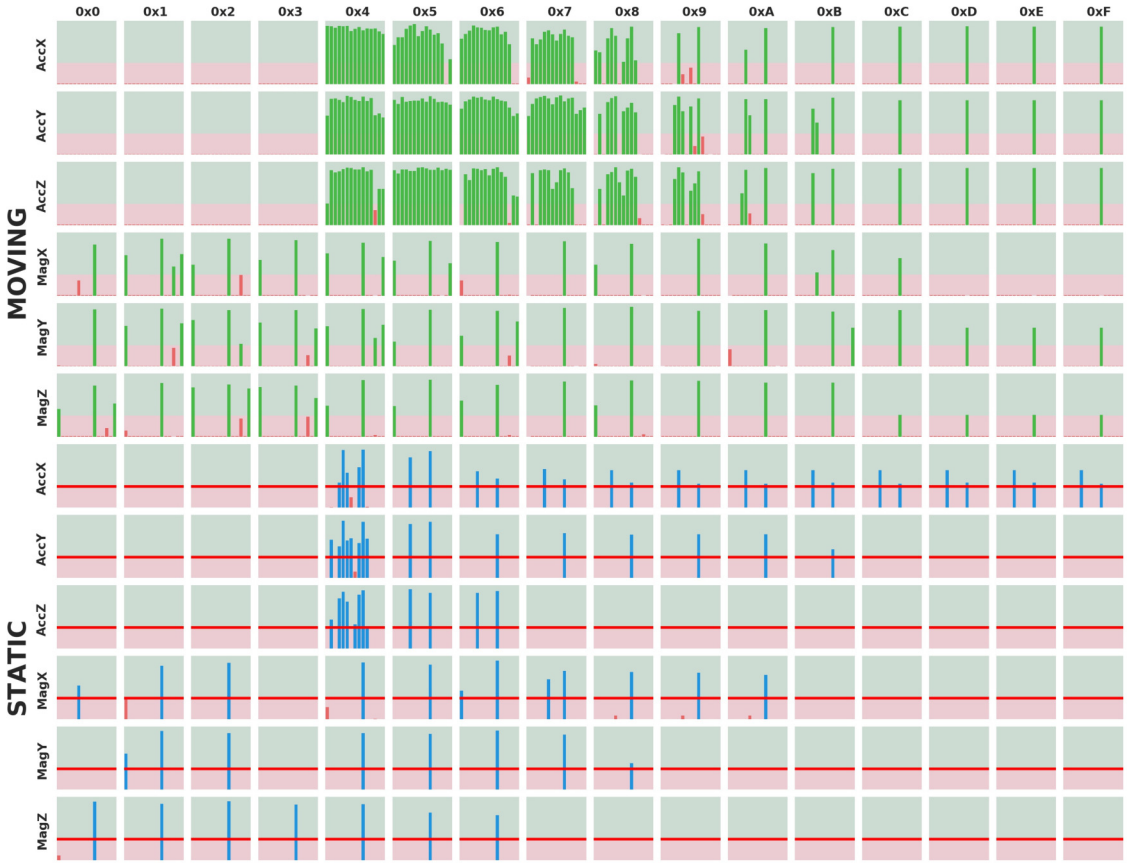

Figure 5.1. Scheme 1 Overview Results

5.3 Scheme 2 Results

The results for Scheme 2, Figures 5.2 and 5.3, demonstrate that this scheme would be an ideal candidate for producing random numbers when moving. Every test is passed handily. Not only would this produce sufficiently random numbers, but because it is the concatenation of already random bits, the speed of this bit stream is three times as fast. The static scenario, however, does not exhibit properties that make it a good candidate for random number generation, as expected.

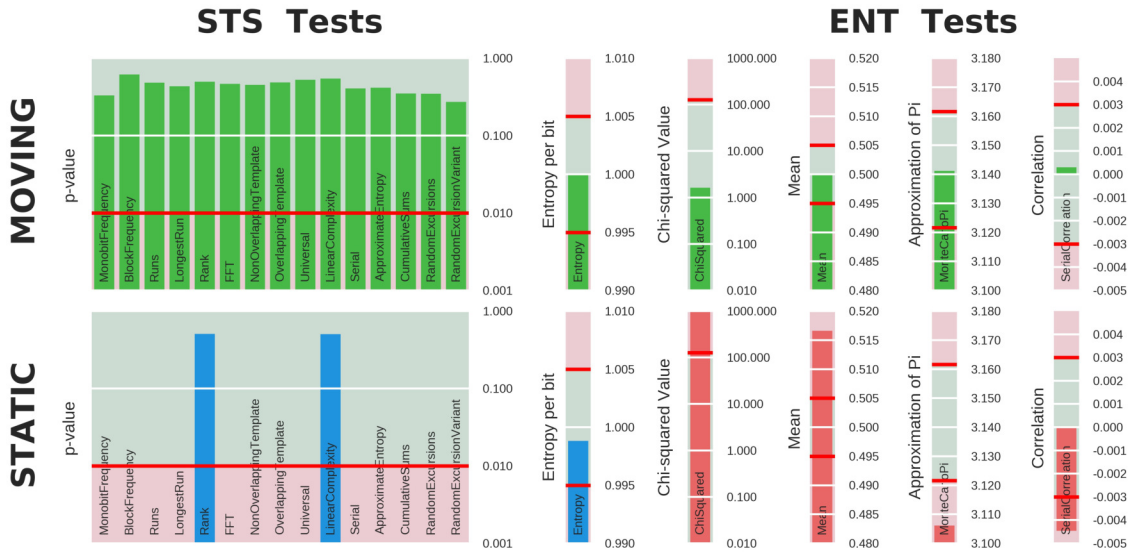

Figure 5.2. Scheme 2 Summary Results

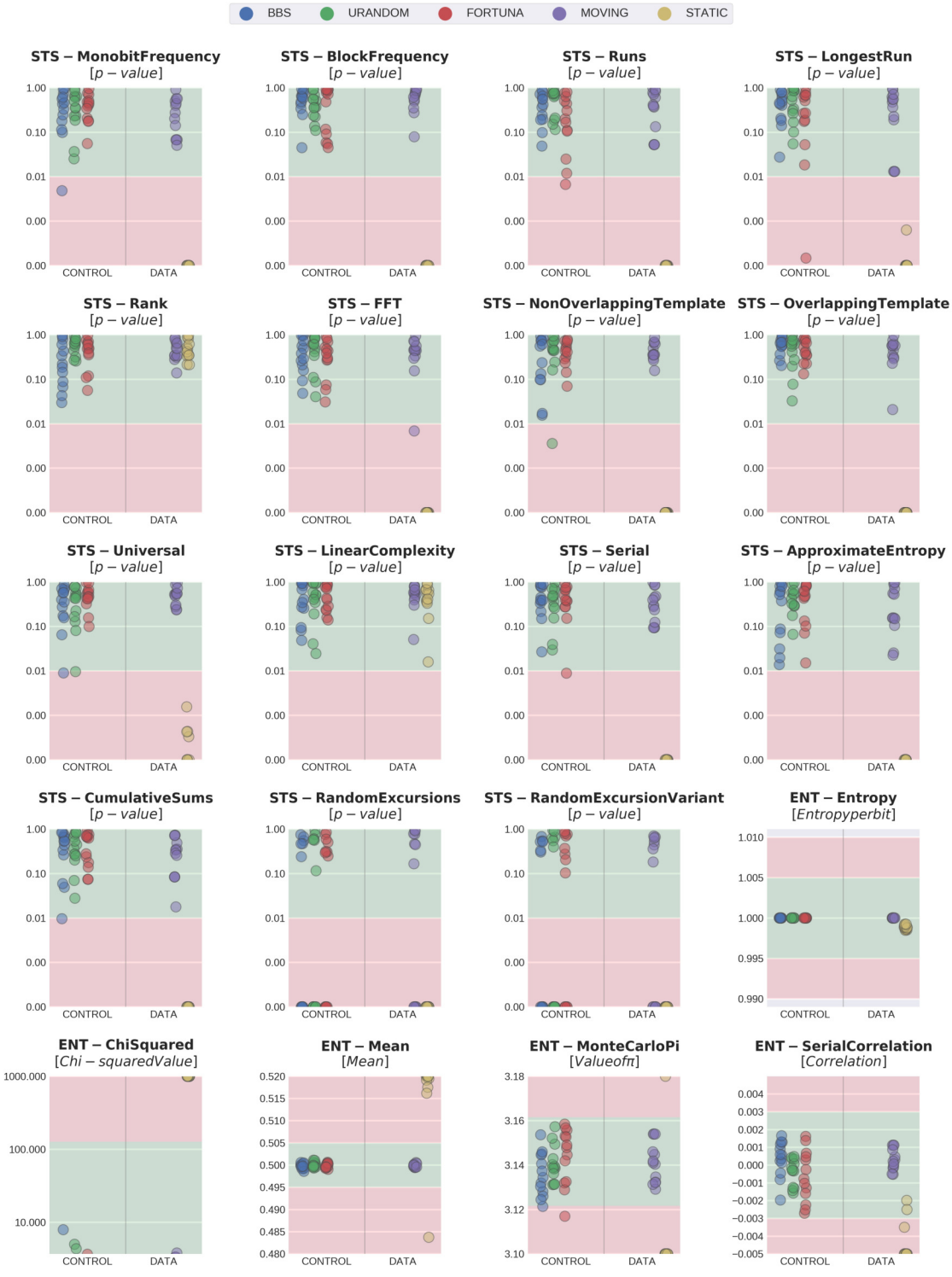

Figure 5.3. Scheme 2 Detailed Results

5.4 Scheme 3 Results

The results for Scheme 3, Figures 5.4 and 5.5, demonstrate for the most part that this scheme would be a decent candidate for producing random numbers when moving. It does, however, fail to pass the STS Random Excursion tests. This is not of too much concern as the STS Random Excursion tests fails frequently [21]. Because we are using multiple bits in this scheme, the amount of bits output is about 1.5 times as fast. This indicates that using a trigger lacking good entropy, in this case the magnetometer LSBs as triggers, does not seem to affect the randomness of the output. The static scenario for Scheme 3, interestingly, seems to pass a number of tests, though not enough to make this scheme a good source of entropy while static.

Figure 5.4. Scheme 3 Summary Results

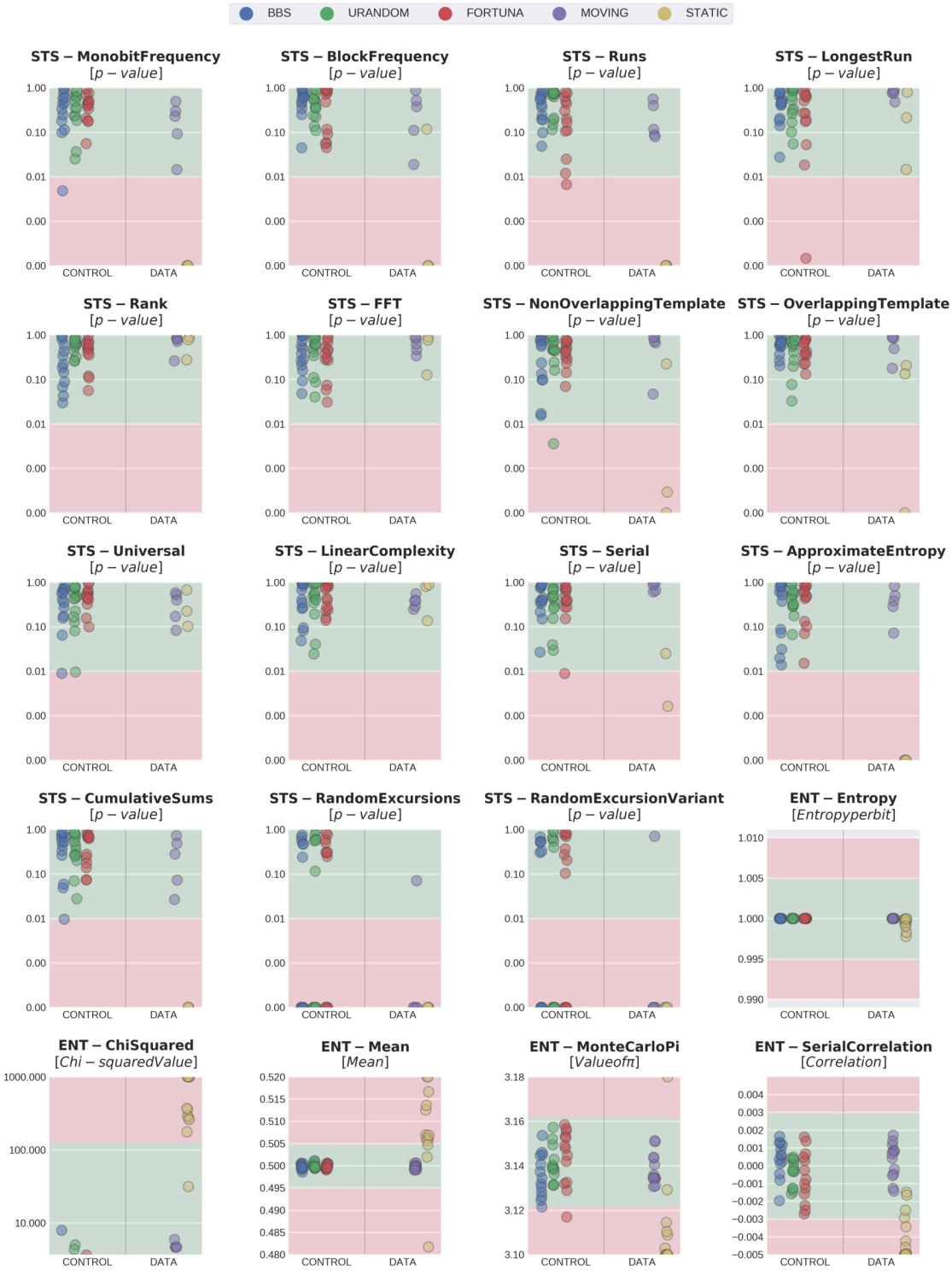

Figure 5.5. Scheme 3 Detailed Results

5.5 Scheme 4 Results

The results for Scheme 4, Figures 5.6 and 5.7, show that while the moving scenario data sets pass a few tests, not enough tests are passed in order for us to confidently use this as a random number generator. An interesting takeaway from this test is that using sufficiently random bits as a trigger, in this case the accelerometer SMRBs, did in fact have an impact on the randomness of the output. The static scenario data sets pass very few tests, making this scheme unfeasible for the static data set.

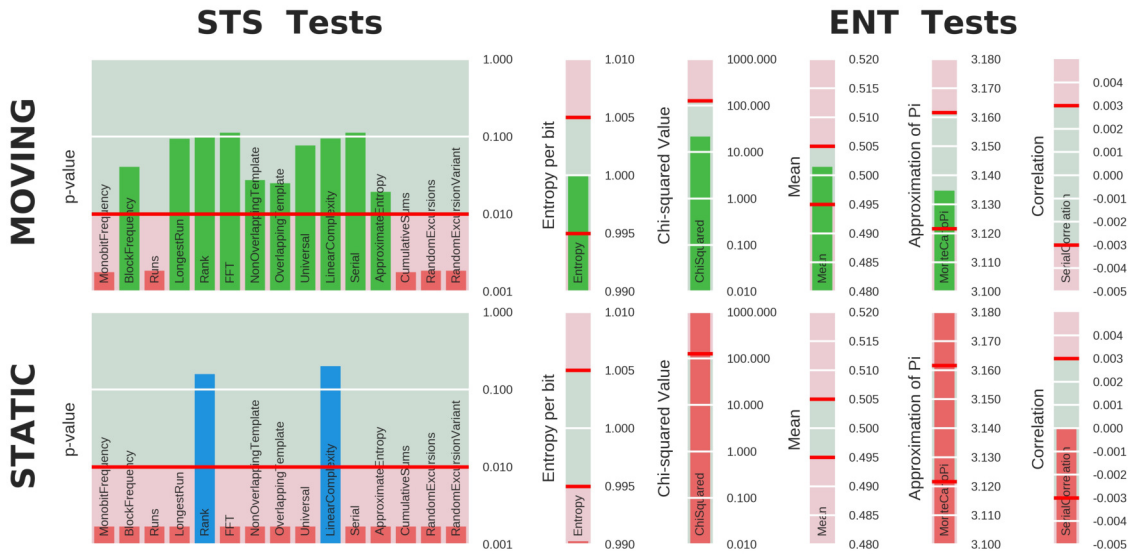

Figure 5.6. Scheme 4 Summary Results

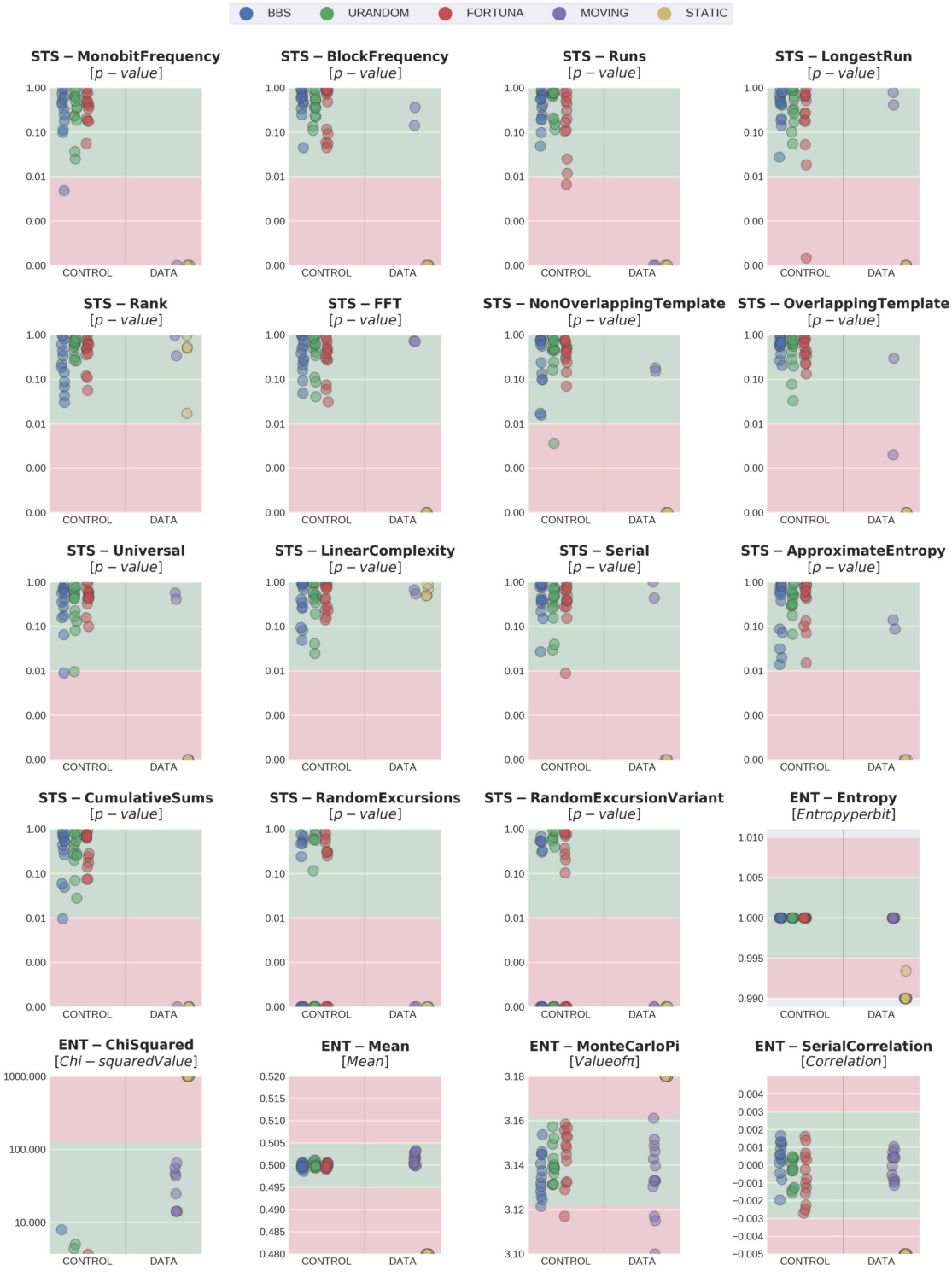

Figure 5.7. Scheme 4 Detailed Results

5.6 Scheme 5 Results

The results for Scheme 5, Figures 5.8 and 5.9, show that using the magnetometer LSBs as a timer does not affect the randomness of the chosen bits very much. This scheme would be a good candidate when using the moving scheme data sets. Unfortunately, this scheme seems to slow down the bit stream by approximately 5%.

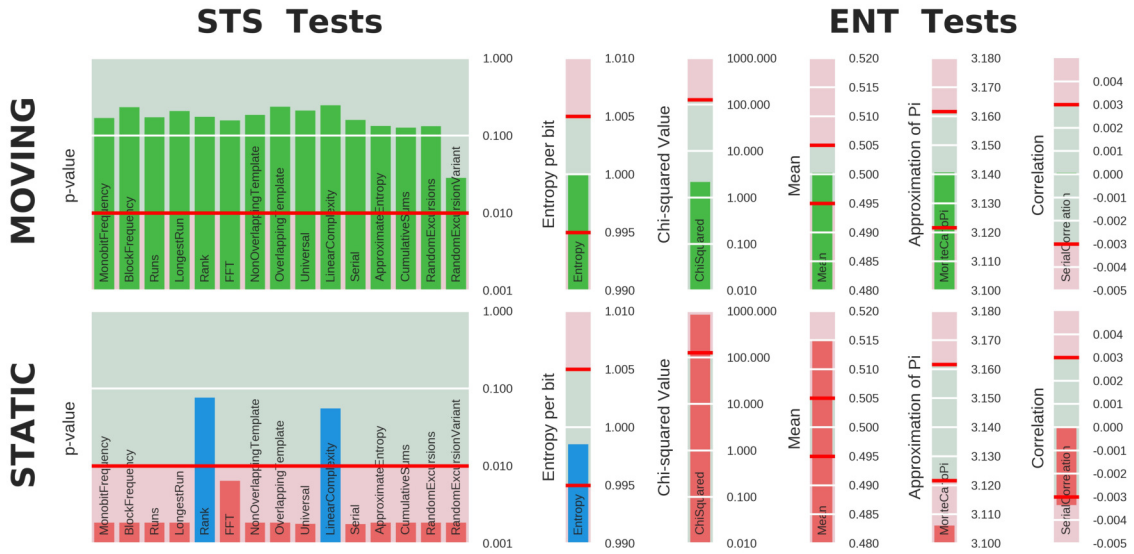

Figure 5.8. Scheme 5 Summary Results

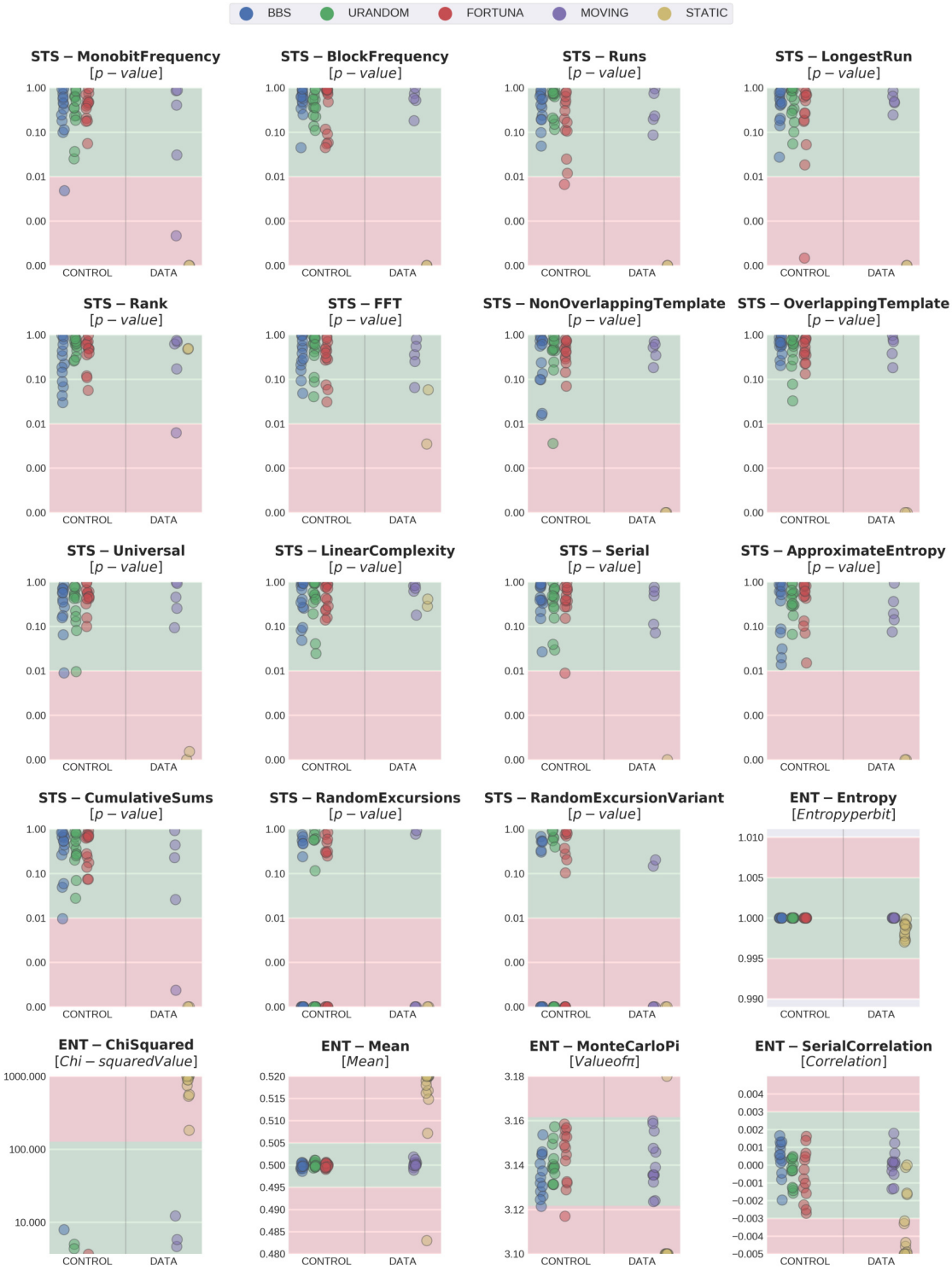

Figure 5.9. Scheme 5 Detailed Results

5.7 Scheme 6 Results

The results for Scheme 6, Figures 5.10 and 5.11, show that using the accelerometer SMRBs as a timer does have an affect on the randomness of the chosen bits. A few of the tests, namely the Block Frequency and FFT tests in the STS suite, pass, but are below the p-value threshold of .1, making them a bit questionable. The static scenario data sets, again, do not demonstrate good properties as a random number generator.

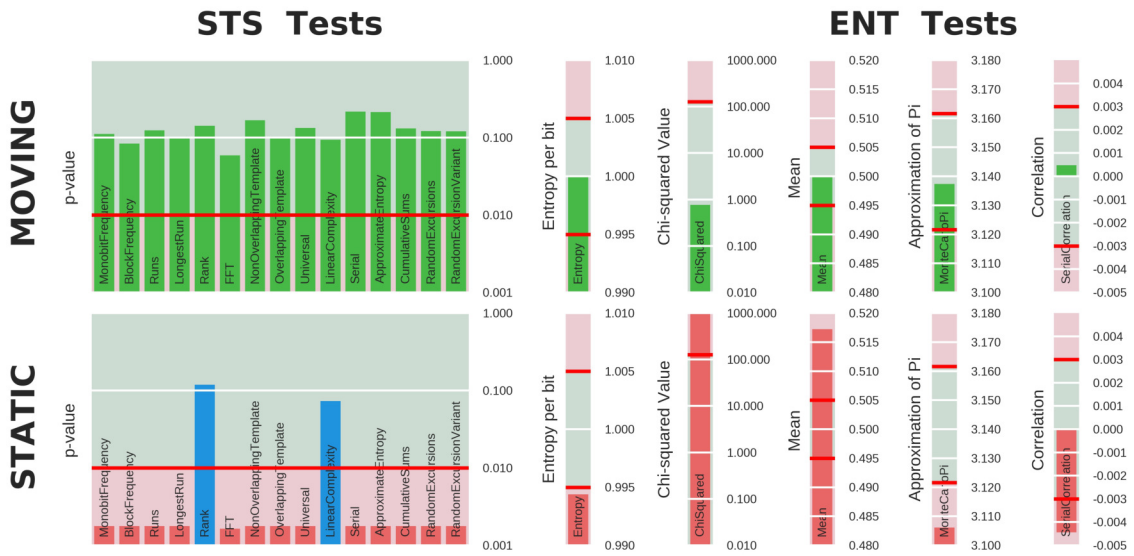

Figure 5.10. Scheme 6 Summary Results

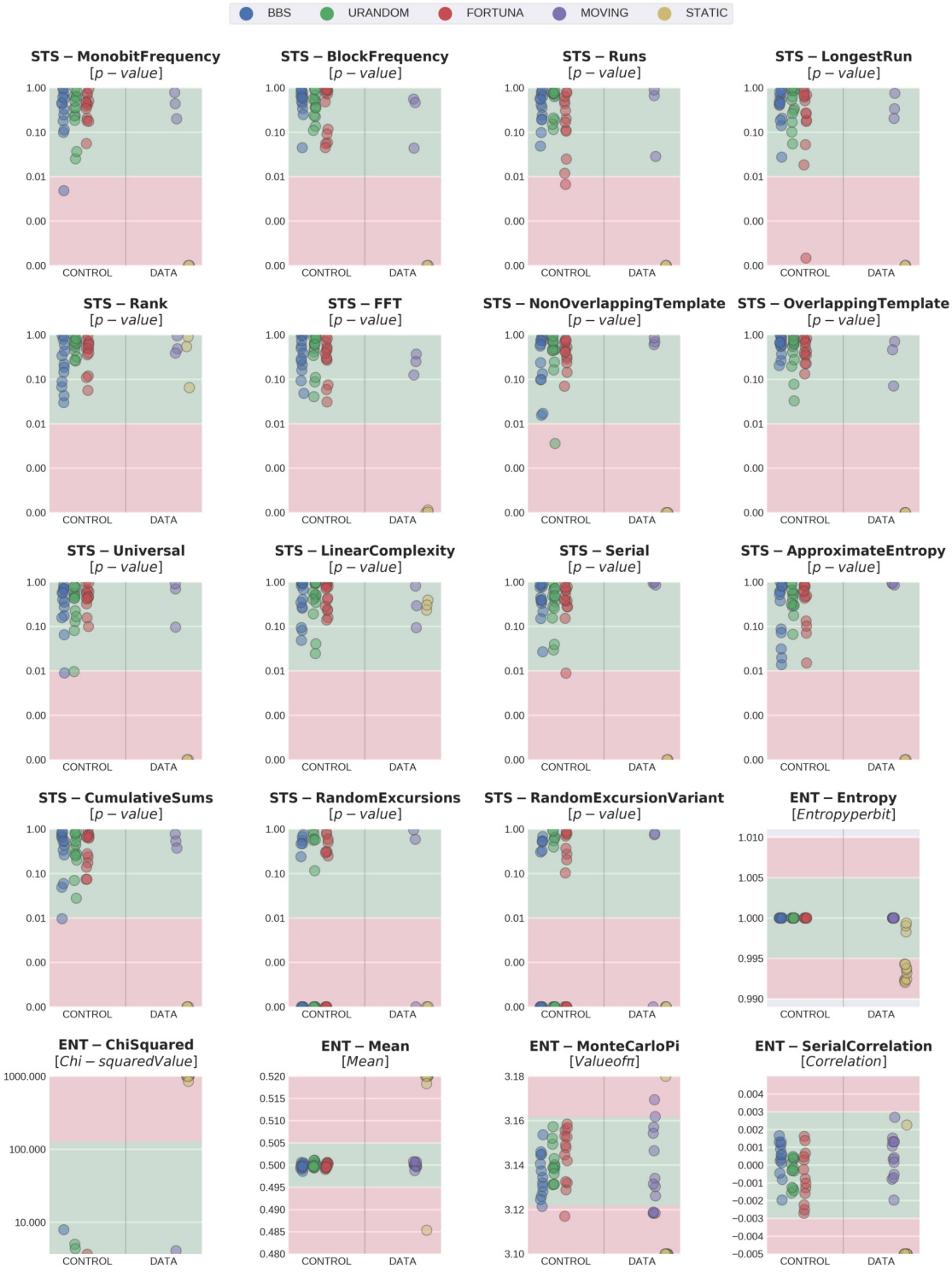

Figure 5.11. Scheme 6 Detailed Results

5.8 Scheme 7 Results

The results for Scheme 7, Figures 5.12 and 5.13, show that using the parity bits of the registers is not a good generation scheme for both the moving and static scenario data sets.

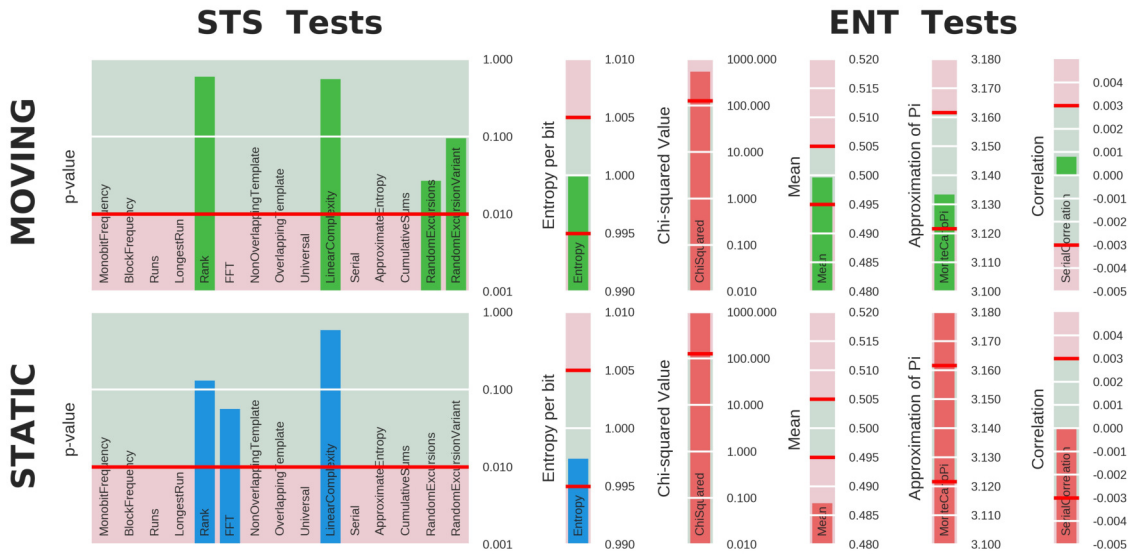

Figure 5.12. Scheme 7 Summary Results

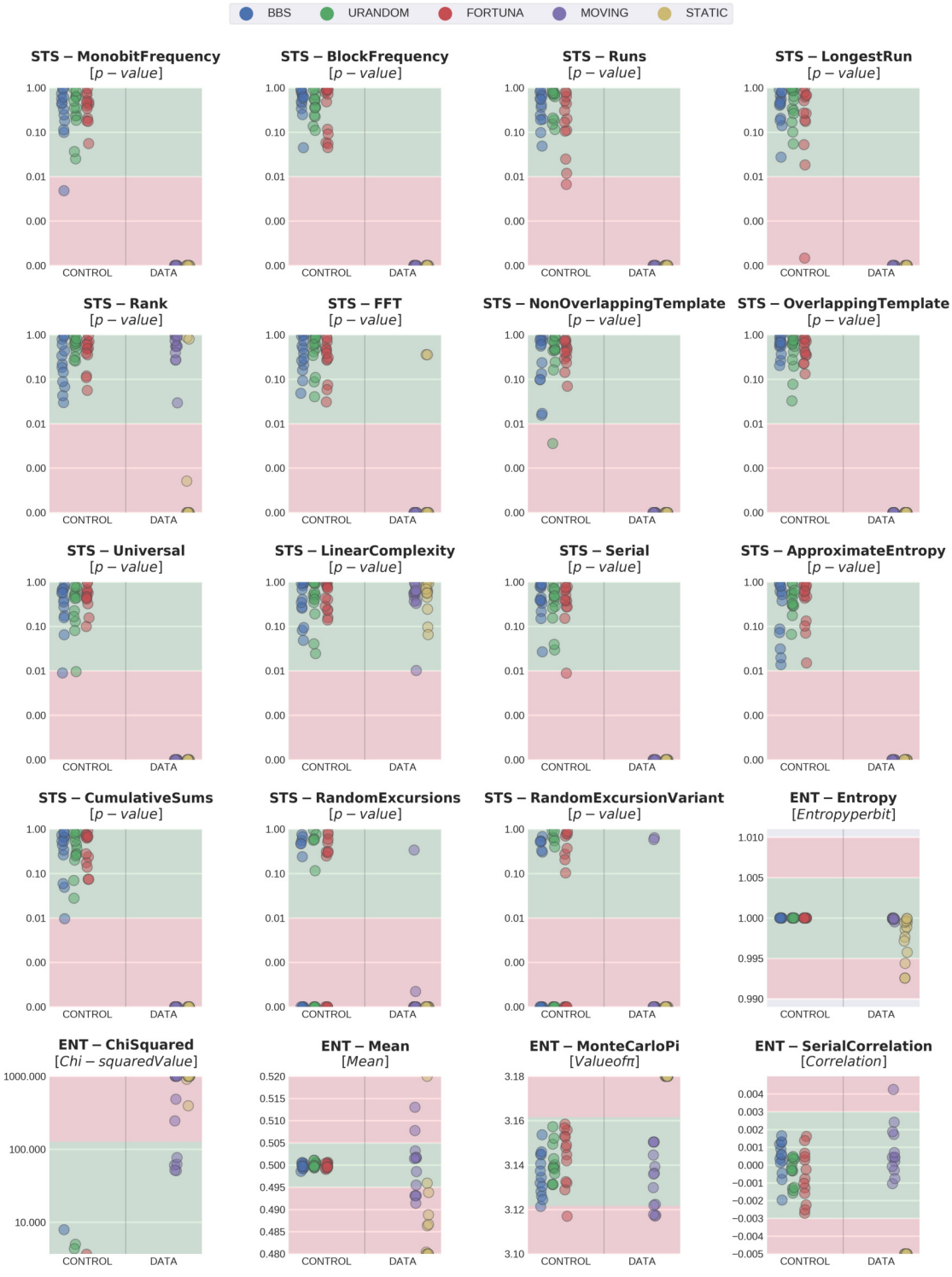

Figure 5.13. Scheme 7 Detailed Results

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Conclusion

In conclusion, the viability of generating true-random numbers from the BMC150 which will be installed on future generations of NPSFS has been demonstrated. There is confidence that further studies would be able to integrate further sources of entropy and implement methods to optimize the sensor input to maximize randomness.

6.1 Findings

We identified specific bits in the BMC150 eCompass that can be used as sources of entropy in and of themselves. We also showed that certain schemes can be used to increase the bit stream speed. The schemes that showed promise are: the concatenation of the identified bits.

6.2 Future Works

This project can lead to further studies in four different areas: what the random numbers that are produced can be used for, what protocols can be used or designed to transmit these random numbers, what other environments are available to use the device, and what other sensors can be used as an entropy source.

6.2.1 Uses for Generated Numbers

Regarding the various uses of these random numbers generated on the NPSFS, one use case it to generate a unique key for the on-board CDMA radio. Another use case would be to broadcast these numbers to other NPSFS in the constellation for use in their computations or key generation. Yet another idea would be to take a page from the GPS satellite playbook and broadcast random numbers down to earth for "the benefit of humanity," streaming continuous random numbers to anyone with a low cost receiver.

6.2.2 Protocols

We see two different categorical methods for using numbers generated by our TRNG. The first is to use a certain number of bits produced as a seed or one-time-used number, or "nonce". In order to do so, one may just collect a certain amount of data without regard to any metadata regarding the data transmission. The second idea would be to have two or more parties establish an a priori time to collect data. In order to communicate these random numbers, various protocols should be studied to determine which method provides the optimal properties when considering transmission speed and data integrity. Another case to consider would be how to describe when one starts and stops collection. This way, people may be able to record and reference specific bit streams to facilitate "sharing" of these random numbers.

6.2.3 Other Environments

Though this study focused on the scenario of the NPSFS operating in free space, further studies could be conducted on how the NPSFS could operate as a TRNG in high altitude, on the surface of the ocean, or even hanging from a stationary object.

6.2.4 Other Sensors to Use as Entropy Sources

Regarding the different sensors that could be tested for randomness on-board the NPSFS, some ideas include: using unshielded memory to determine if any bits are flipped due to cosmic radiation; using radio frequency noise from an on-board radio receiver; and possibly using an on-board camera to obtain images and use the image as a source of randomness.

APPENDIX A:
Rig

A.1 Rig Components

The rig used was made with the following pieces:

- 3D Printer: **Sunhokey Prusa i3**
- Filament Material: **HATCHBOX 3D, PLA 1.75mm, Black**
- Motors: **Upgrade Industries DC 12V 200RPM Mini Metal Gear Motor**
- Motor controllers: **RioRand™ Upgraded 6V-90V 15A DC Motor Controller**
- Bluetooth module: **KEDSUM Arduino HC-06 Serial Bluetooth Module**
- LiPo battery: **Morpilot 3.7V 720mAh 20C Lipo Battery**

A.2 Rig Anatomy

(1) Spinning Arm Controller
(2) Spinning Enclosure Controller
(3) Spinning Arm
(4) Spinning Enclosure Motor
(5) Enclosure
(6) LiPo battery
(7) HC-06 Bluetooth module
(8) Intel Quark D2000

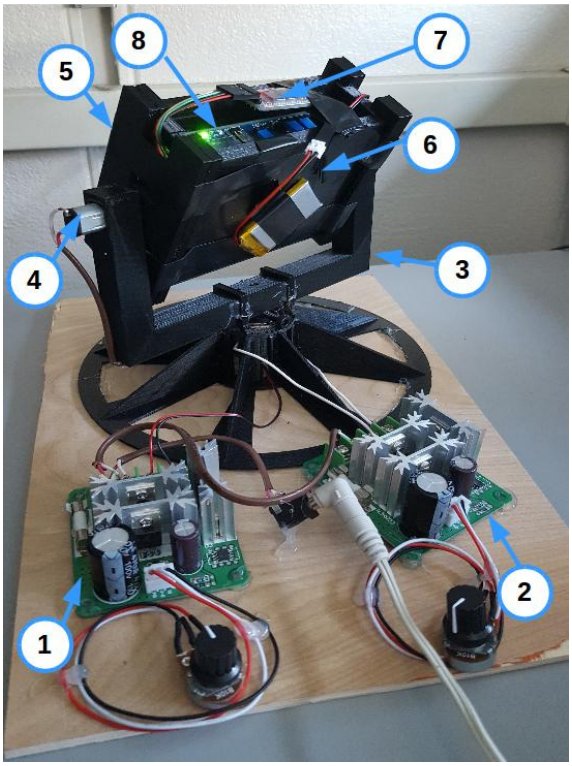

Figure A.1. 3D Motion Simulation Rig

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B:
Supplemental Material

The following is a description of the data provided as supplemental material to this thesis. For access to this supplemental data, please contact the Naval Postgraduate School library.

CAD Files for the 3D Simulation Rig. These are the various 3D files used to produce the 3D simulation rig. The files were created with FreeCAD, an open-source parametric 3D CAD modeler.

Raw Data Collected and Used in Thesis. These are the raw data samples collected and analyzed during the thesis research. Additionally included is the code that was used to produce the BBS and FORTUNA PRNG data used as a control.

Results. These are the results of the experimentation done during this thesis research.

Firmware for Intel Quark D2000. This is the modified library code and firmware flashed onto the Intel Quark D2000 to generate the raw data used. The version of Intel System Studio for Microcontroller used was 3.0.0.20160726 on an ASUS UX501J laptop running Ubuntu 14.04 LTS.

Various Graphics. These are various graphics used in the thesis.

Thesis Presentations. These are the results presentation and NPS Applied Math department presentation. The results presentation was presented after completing the data analysis to Professors Dinolt and Stanica. The final presentation given to the NPS Applied Math department upon completion of the thesis.

Tools. These are various software tools used throughoutt the thesis experimentation.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] K. Saxe. (2008). Mathematics and common sense: A case of creative tension. *The Mathematical Intelligencer*. [Online]. *30*(2). pp. 180–182.

[2] G. J. Chaitin. (1975). Randomness and Mathematical Proof. *Scientific American*. [Online]. *232*(5). pp. 47–52. Available: https://www.cs.auckland.ac.nz/~chaitin/sciamer.html

[3] Entropy (computing). (n.d.). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Entropy_(computing). Accessed Feb. 10, 2017.

[4] N. Samsen. Coin Flipper. [Online]. Available: http://www.dotmancando.info/index.php?/projects/coin-flipper/

[5] M. Haahr. (2017). The History of RANDOM.ORG. [Online]. Available: https://www.random.org/history/

[6] J. Walker. (2017). HotBits: Genuine random numbers, generated by radioactive decay. [Online]. Available: http://www.fourmilab.ch/hotbits/

[7] M. Idrassi. (2017). Random Number Generator. [Online]. Available: https://veracrypt.codeplex.com/wikipage?title=Random%20Number%20Generator

[8] D. E. Knuth, *The Art of Computer Programming*, 3rd ed. Reading, MA: Addison Wesley Longman, 1998.

[9] "/dev/random". *Everything*$_2$. [Online]. Available: http://everything2.com/node/1467767. Accessed Feb 10, 2017.

[10] Z. Manchester. Space Systems Design Studio. [Online]. Available: http://www.spacecraftresearch.com/

[11] Z. Manchester. KickSat – Your personal spacecraft in space! [Online]. Available: https://www.kickstarter.com/projects/zacinaction/kicksat-your-personal-spacecraft-in-space

[12] Z. "Manchester, M. Peck, and A. Filo. KickSat: A Crowd-Funded Mission To Demonstrate The World's Smallest Spacecraft. [Online]. Available: https://zacinaction.github.io/docs/KickSat_SmallSat.pdf

[13] E. Bigham. Next Generation NPS Femto Satellite. [Online]. Available: http://elijahbigham94.wixsite.com/summer16presentation

[14] I. Corporation. Intel Quark Microcontroller D2000. [Online]. Available: http://www.intel.com/content/www/us/en/embedded/products/quark/mcu/d2000/overview.html

[15] R. B. GmbH. BMC150 eCompass. [Online]. Available: https://www.bosch-sensortec.com/bst/products/all_products/homepage_1_ohne_marginalspalte_52

[16] *Intel Quark Microcontroller Software Interface 1.1*, 333612-002EN, Intel Corporation, July 2016, p. 186.

[17] D. Coppersmith, H. Krawczyk, and Y. Mansour, *The Shrinking Generator*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 22–39. Available: http://dx.doi.org/10.1007/3-540-48329-2_3

[18] J. K. M. S. U. Zaman and R. Ghosh. (2012). A Review Study of NIST Statistical Test Suite: Development of an indigenous Computer Package. *CoRR*. [Online]. Available: http://arxiv.org/abs/1208.5740

[19] M. Sys, Z. Riha, V. Matyas, K. Marton, and A. Suciu. (2015). On the Interpretation of Results From the NIST Statistical Test Suite. *Romanian Journal of Information Science and Technology*. [Online]. *18*(1). pp. 18–32.

[20] J. Walker. (2017). HA Pseudorandom Number Sequence Test Program. [Online]. Available: http://www.fourmilab.ch/random/

[21] M. Sys, Z. Riha, V. Matyas, K. Marton, and A. Suciu. (2015). On the Interpretation of Results From the NIST Statistical Test Suite. *Romanian Journal of Information Science and Technology*. [Online]. *18*(1).

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California